A distributed and parallel unite and conquer method to solve sequences of non-Hermitian linear systems

Xinzhe Wu, Serge Petiton

Noname manuscript No.
(will be inserted by the editor)

A Distributed and Parallel Unite and Conquer Method to Solve Sequences of Non-Hermitian Linear Systems

Xinzhe Wu · **Serge G. Petiton**

Received: date / Accepted: date

Abstract Many problems in science and engineering often require to solve a long sequence of large-scale non-Hermitian linear systems with different Right-hand sides (RHSs) but a unique operator. Efficiently solving such problems on extreme-scale platforms requires the minimization of global communications, reduction of synchronization and promotion of asynchronous communications. Unite and Conquer GMRES/LS-ERAM (UCGLE) method [30] is a suitable candidate with the reduction of global communications and the synchronization points of all computing units. It consists of three computing algorithms with asynchronous communications that allow the use of approximate eigenvalues to accelerate the convergence of solving linear systems and to improve the fault tolerance. In this paper, we extend both the mathematical model and the implementation of UCGLE method to adapt to solve sequences of linear systems. The eigenvalues obtained in solving previous linear systems by UCGLE can be recycled, improved on the fly and applied to construct a new initial guess vector for subsequent linear systems, which can achieve a continuous acceleration to solve linear systems in sequence. Numerical experiments using different test matrices to solve sequences of linear systems on supercomputer *Tianhe-2* indicate a substantial decrease in both computation time and iteration steps when the approximate eigenvalues are recycled to generate the initial guess vectors.

Keywords Sequence of linear systems · Linear solvers · Krylov methods · Unite and Conquer · Iterative methods · Eigenvalues

Mathematics Subject Classification (2000) 00A69 · 65F10

This work is funded by the project *MYX* of *French National Research Agency (ANR)* (Grant No. ANR-15-SPPE-003) under the SPPEXA framework.

Xinzhe Wu (✉)
xinzhe.wu.etu@univ-lille.fr
Serge G. Petiton
serge.petiton@univ-lille.fr
Maison de la Simulation, Gif-sur-Yvette Cedex, 91191, France
CRIStAL, University Lille 1, Sciences and Technologies, Villeneuve d'Ascq, 59650, France

1 Introduction

We consider solving a long sequence of sparse and non-Hermitian linear systems

$$Ax = b_i, i = 1,2,3,\cdots \tag{1}$$

where $A \in \mathbf{C}^{n\times n}$ keeps the same, and $b_i \in \mathbf{C}^n$ changing from one system to another. Moreover, these systems are typically not available simultaneously. This kind of linear systems arise from a variety of applications in different scientific and engineering fields, such as the finite element analysis in modeling fatigue [14], diffuse optical tomography [19] and electromagnetic [32], etc. These systems are formatted either by the time-dependent applications where the operator A remains the same, but b_i cannot be obtained simultaneously or by the application of Newton methods for solving nonlinear systems (e.g., [4, 6, 12, 20]).

Different techniques can be applied to reduce the time cost of solving subsequent systems in sequence. If the direct solvers are appropriate, their factorizations can be shared and reused to solve continuous linear systems by forward/backward. If the matrix is sparse with a high dimension, and the direct solvers are not applicable, the iterative methods based on Krylov subspace, such as CG (Conjugate Gradient) method [18] for symmetric systems, and GMRES (Generalized minimal residual) method [26] for non-Hermitian systems, should be considered. In order to accelerate the convergence, the preconditioners can also be applied. For the linear systems with simultaneous RHSs, the block Krylov methods [3, 7, 15, 27] are applicable. For linear systems in sequence with different RHSs, since they share the same operator matrix A, the intermediate information computed from the procedures of solving previous systems can be reused to speed up the solves of subsequent systems. The seed approach [1, 8, 13, 21, 24, 27, 28] selects one seed system and solves it by the Krylov iterative methods. Then, the optimal norm or orthogonality criterion on the Krylov subspace generated by the seed system can be used to form the initial guess vectors of other linear systems. The seed method requires the extra memory space to store the subspace of seed systems, and its speed up cannot always be guaranteed for the uncorrelated right-hand sides. Another alternative is to recycle the Krylov subspace spectral information [5, 17, 19, 22, 32]. The most well-known method GCRO-DR (Generalized Conjugate Residual method with inner Orthogonalization and Deflated Restarting) proposed by Parks et al. [22] allows speeding up the procedure of solving systems from one RHS to another or even between two times restart of iterative methods through the maintenance of the Arnoldi subspace orthogonalization and the deflation of smallest eigenvalues by recycling the approximate invariant subspaces generated by previous solving of linear systems.

However, nowadays, HPC cluster systems continue to scale up not only the number of compute nodes and central processing unit (CPU) cores but also the heterogeneity of components by introducing graphics processing units (GPUs) and manycore processors. This results in the tendency of transition to multi- and many cores within computing nodes, which communicate explicitly through faster interconnection networks. These hierarchical supercomputers can be seen as the intersection of distributed and parallel computing. Indeed, for a large number of cores, the communication of overall reduction operations and global synchronization are the bottleneck [9]. Consequently, large scalar products, overall synchronization, and other operations involving communication among all cores have to be avoided. The numerical methods should be adjusted to reduce the global synchronization points and promote asynchronization to adapt to multi-grain, multi-level memory with high fault tolerance. The special process, e.g., the image projection and sparse matrix-vector product of seed methods, the external solving of eigenvalue problem and reduced QR decomposition

inside GCRO-DR introduce much more additional synchronization points across all cores. These operations damage the performance on extreme-scale platforms. Novel models of algorithms should be proposed to solve linear systems in sequence on large-scale platforms.

UCGLE is a suitable candidate to solve linear systems with single RHS on extreme-scale supercomputing platforms, minimizing global communications and overall synchronization points. Its idea comes from the *unite and conquer approach* introduced by Emad [10] in 2016, which aims at exploring the novel methods for modern computer architectures. It is a model for the design of numerical methods by combining different computing components with asynchronous communication. Moreover, these components work for a same objective. In the *unite and conquer* methods, different parallel computing components work independently, which can be deployed on various platforms such as P2P, cloud, and the supercomputer systems, or different computing units on the same platform. However, UCGLE that we talked about in this paper is only an implementation of *unite and conquer approach* targeting at the modern supercomputing platforms. UCGLE is a hybrid method which consists of three computing components with asynchronous communications: ERAM (Explicitly Restarted Arnoldi Method), GMRES and LS (Least Squares) components. GMRES Component is used to solve the systems, and LS Component performs as a preconditioner using the eigenvalues approximated by ERAM Component to speed up the convergence. LS Component is implemented based on the Least Squares polynomial method proposed by Saad [23]. Compared with the classical hybrid methods using LS polynomial [11, 16, 33], the key features of UCGLE are its distributed and parallel asynchronous communications and the manager engine implementation among three components, specified for supercomputing platforms. Its asynchronous communications cover overall synchronization overhead and improve the fault tolerance. Better convergence acceleration and parallel performance of UCGLE using test matrices compared with preconditioned GMRES are given in [30].

Despite the advantages of UCGLE for large-scale platforms, it is not able to solve sequences of linear systems with continuous improvement through the maintenance of intermediate information. In this paper, we develop a variant of UCGLE to solve the linear systems in sequence by recycling the dominant eigenvalues. Indeed, when solving linear systems with UCGLE, more eigenvalues are approximated, and the more accurate they are, the more significant the acceleration. The eigenvalues obtained from the solves of previous systems can be reused and improved on the fly, to speed up the solves of successive systems. More importantly, these previously calculated eigenvalues can be applied to construct a new initial guess for current system using the LS polynomial method, and its convergence can be significantly accelerated. In this paper, we assume that the matrix A keeps the same for linear systems in sequence, but UCGLE is applicable even part of their spectra change slowly.

The paper is organized as follows. In Section 2, we summarize the theory of numerical algorithms inside UCGLE, especially the LS polynomial method proposed by Saad. The introduction of LS polynomial method shows the fundamentals of recycling the eigenvalues. The extended implementation of UCGLE for solving sequences of linear systems is shown in Section 3. In Section 4, various experiments using UCGLE to solve linear systems in the sequence are presented. We give the conclusions and perspectives in Section 5.

2 UCGLE: Algorithms of Components

In order to give a glance at UCGLE, especially the way to recycle the eigenvalues to solve sequences of linear systems, we summarize the mathematical models and algorithms of three components inside UCGLE in this section.

GMRES and ERAM algorithms inside UCGLE are both Krylov subspace methods. In linear algebra, the m-order Krylov subspace generated by a $n \times n$ matrix A and a vector b of dimension n is the linear subspace spanned as:

$$K_m(A,b) = span(b, Ab, A^2b, \cdots, A^{m-1}b) \tag{2}$$

Arnoldi reduction is able to build an orthogonal basis of K_m. As shown in Algorithm 1, at each step of Arnoldi reduction, it multiplies the previous Arnoldi vector v_j by matrix A, and get an orthogonal vector v_{j+1} against all previous ω_i by a stand Gram-Schmit procedure. It will stop if the vector computed in line 7 is zero. The vectors $\omega_1, \omega_2 \cdots, \omega_m$ form an orthonormal basis of the Krylov subspace.

Algorithm 1 Arnoldi Reduction

```
1: function AR(input:A, m, v, output: H_m, V_m)
2:     v_1 = v/||v||_2
3:     for j = 1, 2, ..., m do
4:         h_{i,j} = (A v_j, v_i), for i = 1, 2, ..., j
5:         ω_j = A v_j − Σ_{i=1}^{j} h_{i,j} v_i
6:         h_{j+1,j} = ||ω_j||_2
7:         if h_{j+1,j} = 0 then Stop
8:         end if
9:         v_{j+1} = ω_j / h_{j+1,j}
10:    end for
11: end function
```

Denote by V_m, the $n \times m$ matrix with column vectors $v_1, v_2 \cdots, v_m$, by $\overline{H}_m$, the $(m+1) \times m$ Hessemberg matrix whose nonzero entries $h_{i,j}$ are defined by Algorithm 1, then note H_m as the matrix obtained from $\overline{H}_m$ by deleting its last row. The following relations are given:

$$V_m^T A V_m = H_m. \tag{3}$$

In case that the norm of ω_j in line 7 of Algorithm 1 vanishes at a certain step j, H_m turns to be H_j with dimension $j \times j$. The ERAM and GMRES components of UCGLE are both the iterative methods based on the Arnoldi reduction.

2.1 ERAM Algorithm

The mission of ERAM Component inside UCGLE is to approximate the dominant eigenvalues using ERAM, which is a variant of Arnoldi algorithm [2]. Arnoldi method is widely used to compute the eigenvalues of large sparse non-Hermitian matrices. Its kernel is the Arnoldi reduction, which gives an orthonormal basis $V_m = (v_1, v_2, \cdots, v_m)$ of $K_m(A, v)$, where A is $n \times n$ matrix, and v is a n-dimensional vector. With the relation (3), the eigenvalues of H_m are the approximated ones of A, which are called the Ritz values of A. The r desired Ritz values $\Lambda_r = (\lambda_1, \lambda_2, \cdots, \lambda_r)$ can be gotten by basic Arnoldi method.

The numerical accuracy of the computed eigenpairs by basic Arnoldi method depends highly on the size of $K_m(A, v)$ and the orthogonality of V_m. Generally, the larger the subspace is, the better the eigenpairs approximation is. The problem is that firstly the orthogonality of the computed V_m tends to degrade with each basis extension. Also, the larger the subspace size is, the larger the V_m matrix gets. Hence available memory may also limit the subspace

size, and so the achievable accuracy of the Arnoldi process. To overcome this, Saad [25] proposed to restart the Arnoldi process, which is the ERAM. The Krylov subspace size of ERAM is fixed as m, and only the starting vector will vary. After one restart of the Arnoldi process, the starting vector will be initialized using information from the computed Ritz vectors. Then the vector will be forced to be in the desired invariant subspace. The Algorithm of ERAM is given by Algorithm 2, where ε_a is a tolerance value, r is desired eigenvalues number and the function g defines the stopping criterion of iterations.

Algorithm 2 Explicitly Restarted Arnoldi Method

function ERAM($input$: $A, r, m, v, \varepsilon_a$, $output$: Λ_r)
  Compute an AR($input$:A, m, v, $output$: H_m, V_m)
  Compute r desired eigenvalues λ_i ($i \in [1, r]$) of H_m
  Set $u_i = V_m y_i$, for $i = 1, 2, \cdots, r$, the Ritz vectors
  Compute $R_r = (\rho_1 . \cdots, \rho_r)$ with $\rho_i = ||\lambda_i u_i - A u_i||_2$
  if $g(\rho_i) < \varepsilon_a$ ($i \in [1, r]$) **then**
    stop
  else
    set $v = \sum_{i=1}^{d} Re(v_i)$, and GOTO 2
  end if
end function

2.2 GMRES Algorithm

GMRES is a Krylov iterative method to solve non-Hermitian linear systems $Ax = b$. It approximates the solution x_m from an initial guess x_0, with the minimal residual in a selected Krylov subspace. It was introduced by Saad and Schultz in 1986 [26].

Algorithm 3 Basic GMRES method

function BASICGMRES($input$: A, m, x_0, b, $output$: x_m)
  $r_0 = b - Ax_0, \beta = ||r_0||_2$, and $v_1 = r_0/\beta$
  Compute an AR($input$:A, m, v_1, $output$: H_m, V_m)
  Compute y_m which minimizes $||\beta e_1 - H_m y||_2$
  $x_m = x_0 + V_m y_m$
end function

In fact, any vector x in subspace $x_0 + K_m$ can be written as

$$x = x_0 + V_m y. \tag{4}$$

with y an m-vector, Ω_m an orthonormal basis of the Krylov subspace K_m. The norm of residual $R(y)$ of $Ax = b$ is given as:

$$\begin{aligned} R(y) &= ||b - Ax||_2 = ||b - A(x_0 + V_m y)||_2 \\ &= ||V_{M+1}(\beta e_i - \overline{H}_m y)||_2 = ||\beta e_i - \overline{H}_m y||_2. \end{aligned} \tag{5}$$

x_m can be obtained as $x_m = x_0 + V_m y_m$ where $y_m = argmin_y ||\beta e_i - \overline{H}_m y||_2$. The minimizer y_m is inexpensive to compute from this least-squares problem if m is typically small. This

is the basic GMRES method as Algorithm 3. If m is large, GMRES can be restarted after a number of iterations, to avoid large memory and computational requirements with the increase of Krylov subspace projection number. The difficulty of restarted GMRES is that it can stagnate when the matrix is not positive definite. Usually, the preconditioning techniques or the deflation of eigenpairs are used to speed up the converge.

2.3 Least Squares Polynomial Algorithm

LS polynomial method is an iterative method proposed by Saad [23] in 1987 to solve linear systems. In this paper, it aims to calculate a new preconditioned residual for the restart of GMRES and to generate the initial guess vectors for solving the sequences of linear systems.

2.3.1 Polynomial Iterative Method

The iterates of LS polynomial method can be written as

$$x_d = x_0 + P_d(A)r_0. \tag{6}$$

where x_0 is an initial approximation of solution, r_0 the corresponding residual norm, and P_d a polynomial of degree $d-1$. We set a polynomial R_d of degree d such that

$$R_d(\lambda) = 1 - \lambda P_d(\lambda). \tag{7}$$

The residual of d^{th} steps iteration r_d can be expressed as equation

$$r_d = R_d(A)r_0. \tag{8}$$

with the constraint $R_d(0) = 1$. We want to find a kind of polynomial which can minimize $||R_d(A)r_0||_2$, with $||.||_2$ the Euclidean norm.

If A is a $n \times n$ diagonalizable matrix with its spectrum denoted as $\sigma(A) = \lambda_1, \cdots, \lambda_n$, and the associated eigenvectors $u_1, \cdots, u_n$. Expanding the initial residual vector r_0 in the basis of these eigenvectors as

$$r_0 = \sum_{i=1}^{n} \rho_i u_i \tag{9}$$

and then the residual vector r_d can be expanded in this basis of eigenvectors as

$$r_d = \sum_{i=1}^{n} R_d(\lambda_i)\rho_i u_i \tag{10}$$

which allows getting the upper limit of $||r_d||$ as

$$||r_d||_2 \leq ||r_0||_2 \max_{\lambda \in \sigma(A)} |R_d(\lambda)| \tag{11}$$

In order to minimize the norm of r_d, it is possible to find a polynomial P_d which can minimize the Equation (11). And it tends to be a minimum-maximum problem with the constraint $R_d(0) = 1$ and $\lambda \in \sigma(A)$

$$min\ max_{\lambda \in \sigma(A)} |R_d(\lambda)| \tag{12}$$

In the practical situation, when solving the linear systems by Least Squares Method, the whole number of eigenvalues λ_i of A are usually not available. But a region H in the real complex plane that includes $\lambda(A)$ can be constructed by the approximative eigenvalues of A, and then the problem becomes as below with $R_d(0) = 1$

$$min\ max_{\lambda \in H} |R_d(\lambda)| \tag{13}$$

2.3.2 Least Squares Polynomial Preconditioning

Indeed, it is not always easy to approximate all the eigenvalues of A. Suppose that the total number of eigenvalues $\sigma(A)$ can be divided into $\sigma_k(A) = \lambda_1, \cdots . \lambda_k$ and $\sigma_{n-k}(A) = \lambda_{k+1}, \cdots, \lambda_n$, where σ_k are the first k known dominant eigenvalues, and σ_{n-k} are the rest unknown ones. Thus the Equation (9) can be rewritten as

$$r_0 = \sum_{i=1}^{k} \rho_i u_i + \sum_{i=k+1}^{n} \rho_i u_i \tag{14}$$

And the Equation (10) can be decomposed as

$$r_d = \sum_{i=1}^{k} R_d(\lambda_i) \rho_i u_i + \sum_{i=k+1}^{n} R_d(\lambda_i) \rho_i u_i \tag{15}$$

Note H_k as the convex hull of related first k dominant eigenvalues. The minimum-maximum problem with H_k and constraint $R_d(0) = 1$ becomes as below

$$min\ max_{\lambda \in H_k} |R_d(\lambda)| \tag{16}$$

2.3.3 Calcul the Iteration Form

In order to solve this minimum-maximum problem, a well known method is to use the Chebyshev polynomial, where H_k is taken to be an ellipse with center c and focal distance e, which contains the convex hull of $\sigma_k(A)$. If the origin is outside of it, the minimal polynomial can be reduced to a scaled and shifed Chebyshev polynomial:

$$R_d(\lambda) = \frac{T_d(\frac{c-\lambda}{e})}{T_d(\frac{c}{e})} \tag{17}$$

The three terms recurrence of Chebyshev polynomial introduces an elegant algorithm for generating the approximation x_d that uses only three vectors of storage. The choice of ellipses as enclosing regions in Chebyshev acceleration may be overly restrictive and ineffective if the shape of the convex hull of the unwanted eigenvalues bears little resemblance to an ellipse. There are various research to find the acceleration polynomial to minimize its L_2-norm on the boundary of the convex hull of the unwanted eigenvalues with respect to some suitable weight function ω. An algorithm based on the modified moments for computing the least square polynomial was proposed by Youcef Saad [23]. The problem tends to find a polynomial P_d on the boundary of H_k which we note as ∂H_k, that maximizes the modulus of $|1 - \lambda P_d(\lambda)|$. And then we get the least square problem with respect to some weight $w(\lambda)$ on the boundary of H_k and the constraint $R_d(0) = 1$.

Algorithm 4 Least Squares Polynomial Pre-treatement

function LS($input$: A, b, d, Λ_r, $output$: A_d, B_d, Δ_d, H_d)
  construct the convex hull C by Λ_r
  construct $ellispe(a,c,e)$ by the convex hull C
  compute parameters A_d, B_d, Δ_d by $ellispe(a,c,e)$
  construct matrix T $(d+1) \times d$ matrix by A_d, B_d, Δ_d
  construct Gram matrix M_d by Chebyshev polynomials basis
  Cholesky factorization $M_d = LL^T$
  $F_d = L^T T$
  H_d satisfies min $\|l_{11}e_1 - F_d H\|$
end function

The iteration form of Least Squares is $x_d = x_0 + P_d(A)r_0$ with P_d the least square polynomial of degree $d-1$ under the Formula (18). The polynomial basis t_i meet the three terms recurrence relation (19).

$$P_d = \sum_{i=0}^{d-1} \eta_i t_i. \tag{18}$$

$$t_{i+1}(\lambda) = \frac{1}{\beta i + 1}[\lambda t_i(\lambda) - \alpha_i t_i(\lambda) - \delta_i t_{i-1}] \tag{19}$$

For the computation of parameters $H = (\eta_0, \eta_1, \cdots, \eta_{d-1})$, a modified Gram matrix M_d with dimension $d \times d$ and a matrix T_d with dimension $(d+1) \times d$ are constructed by the three terms recurrence of the basis t_i. M_d can be factorized to be $M_d = LL^T$ by the Cholesky factorization. The parameters H can be computed by a least squares problem of the formula

$$min\|l_{11}e_1 - F_d H\| \tag{20}$$

We need compute the vectors $\omega_i = t_i(A)r_0$, and get the linear combination of (18). The recurrence expression of ω_i is given as (21) and the final solution as (22).

$$\omega_{i+1} = \frac{1}{\beta_{i+1}}(A\omega_i - \alpha_i\omega_i - \delta_i\omega_{i-1}) \tag{21}$$

$$x_d = x_0 + P_d(A)r_0 = x_0 + \sum_{i=1}^{d-1} \eta_i \omega_i \tag{22}$$

Algorithm 5 Update GMRES residual by LS Polynomial

function LSUPDATERESIDUAL($input$:$A, b, A_d, B_d, \Delta_d, H_d$)
  $r_0 = b - Ax_0$, $\omega_1 = r_0$ and $r_0 = 0$
  for $k = 1, 2, \cdots, s_{use}$ **do**
    for $i = 1, 2, \cdots, d-1$ **do**
      $\omega_{i+1} = \frac{1}{\beta_{i+1}}[A\omega_i - \alpha_i\omega_i - \delta_i\omega_{i-1}]$
      $x_{i+1} = x_i + \eta_{i+1}\omega_{i+1}$
    end for
  end for
  Update GMRES restarted residual by x_d
end function

2.3.4 Algorithm

The pseudocode of this method is presented in Algorithm 4, where A is a $n \times n$ matrix, b is a right-hand vector of dimension n, d is the degree of Least Squares polynomial, Λ_r the collection of approximate eigenvalues, and the output values are $A_d = (\alpha_0, \alpha_1, \cdots, \alpha_{d-1})$, $B_d = (\beta_1, \beta_2, \cdots, \beta_d)$, $\Delta_d = (\delta_1, \delta_2, \cdots, \delta_{d-1})$, and $H_d = (\eta_0, \eta_1, \cdots, \eta_{d-1})$, which will be used for constitution of a new GMRES initial vector. a, c, e are the required parameters to fix an ellipse in the plan, with a the distance between the vertex and centre, c the centre position and e the focal distance. The pre-treatement procedure of LS polynomial method to generate these parameters is given in Algorithm 4. The iteration form to generate the LS preconditioned residual for GMRES Component is also shown in Algorithm 5, which takes place inside GMRES Component of UCGLE for the pratical implementation. In this algorithm, x_0 is the temporary solution in GMRES before performing the restart.

3 UCGLE for Sequences of Linear Systems

Based on the summary of GMRES, ERAM and LS algorithms of UCGLE, in this section, firstly we summarize the distributed and parallel implementation of UCGLE with asynchronous communications, then analyze the relation of eigenvalues approximated by ERAM and the convergence speedup of GMRES Component through LS polynomial residuals. This analysis allows us to develop the variant of UCGLE to solve sequences of non-Hermitian linear systems.

3.1 Workflow and Component Implementation

UCGLE composes mainly two parts: the first part uses the restarted GMRES method to solve the linear systems; in the second part, it computes a specific number of approximated eigenvalues, and then applies them to the Least Squares method and gets a new preconditioned residual, as a new initial vector for restarted GMRES.

Fig. 1 gives the workflow of UCGLE and Algorithm 6 shows its implementation of three components with asynchronous communications. ERAM and GMRES components are implemented in parallel, LS Component is in serial. ERAM Component computes a desired number of eigenvalues, and then sends them to LS Component; LS Component uses these received eigenvalues to generate the LS pre-treatment parameters, and sends them to GMRES Component; GMRES Component uses these parameters to generate a new vector as a new restarted vector for solving linear systems. One characteristic of UCGLE is its engine to manger different components and their asynchronous communications. This engine is implemented based on MPI. UCGLE has three levels of parallelism, which is suitable for the architecture of modern large-scale platforms. The Coarse Grain/Component level, Medium Grain/inter-component level, and fine Grain/Thread level (such as GPUs and OpenMP threads) of parallelism were shown in [30].

3.2 Relation between LS Residual and Approximate Eigenvalues

Suppose that the computed convex hull by Least Squares contains eigenvalues $\lambda_1, \cdots, \lambda_m$, the residual given by Least Squares polynomial of degree $d-1$ is

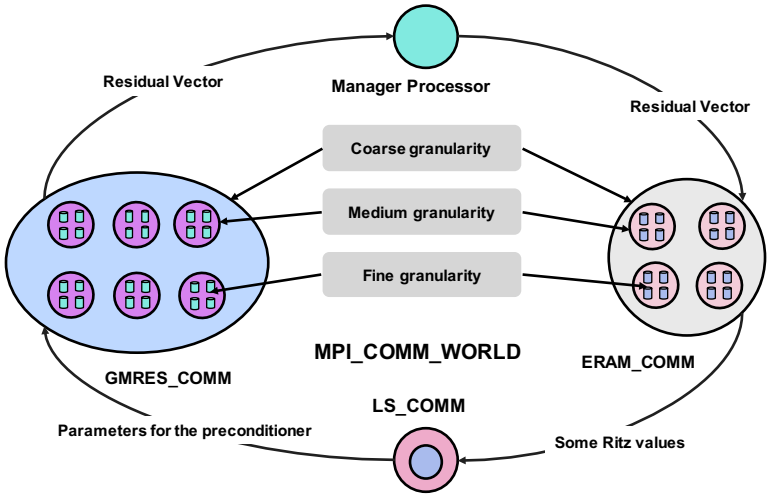

Fig. 1 Workflow, asynchronous communication and different levels parallelism of UCGLE. [30]

$$r = \sum_{i=1}^{k} \rho_i(R_d(\lambda_i))^l u_i + \sum_{i=m+1}^{n} \rho_i(R_d(\lambda_i))^l u_i, \tag{23}$$

this residual can be divided into two parts. The first part is formulated with the first known m eigenvalues which are used to computed the convex hull by LS Component, the second part represents the residual with unknown eigenpairs.

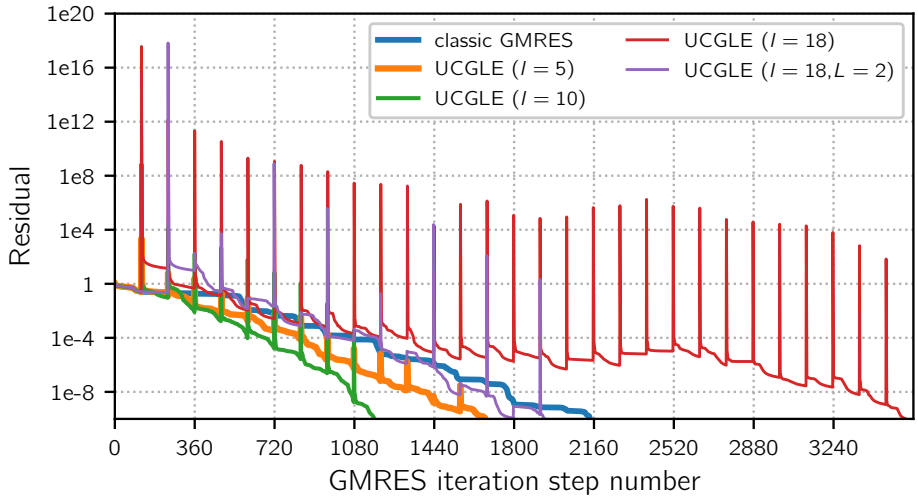

Fig. 2 Convergence comparison of UCGLE method vs classic GMRES.

In practice, for each time preconditioning by LS polynomial method, it is often repeated for several times to improve its acceleration of convergence, that is the meaning of parameter

Algorithm 6 Implementation of Components [30]

function LOADERAM($input$: $A, m_a, \nu, r, \varepsilon_a$)
  while exit==False **do**
    ERAM($A, r, m_a, \nu, \varepsilon_a$, $output$: Λ_r)
    Send (Λ_r) to LS
    if $savefl g == TRUE$ **then**
      write (Λ_r) to file *eigenvalues.bin*
    end if
    if Recv (X_TMP) **then**
      update X_TMP
    end if
    if Recv ($exit == TRUE$) **then**
      Send ($exit$) to LS Component
      stop
    end if
  end while
end function
function LOADLS($input$: A, b, d)
  if Recv(Λ_r) **then**
    LS($input$: A, b, d, Λ_r, $output$: A_d, B_d, Δ_d, H_d)
    Send (A_d, B_d, Δ_d, H_d) to GMRES Component
  end if
  if Recv ($exit == TRUE$) **then**
    stop
  end if
end function
function LOADGMRES($input$: $A, m_g, x_0, b, \varepsilon_g, L, l$, $output$: x_m)
  $count = 0$
  BASICGMRES($input$: A, m, x_0, b, $output$: x_m)
  $X\_TMP = x_m$
  Send (X_TMP) to ERAM Component
  if $||b - Ax_m|| < \varepsilon_g$ **then**
    return x_m
    Send ($exit == TRUE$) to ERAM Component
    Stop
  else
    if $count \mid L$ **then**
      if recv (A_d, B_d, Δ_d, H_d) **then**
        LSUpdateResidual($input$: A, b, B_d, Δ_d, H_d)
        $count++$
      end if
    else
      set $x_0 = x_m$, and GOTO 1
      $count++$
    end if
  end if
  if Recv ($exit == TRUE$) **then**
    stop
  end if
end function

l in Equation (23). The LS polynomial preconditioning applies r_d as a deflation vector for each time GMRES restart process. Fig. 2 gives the comparison between classic GMRES and UCGLE with different values for the parameter l. As shown in this figure, the first part in Equation (23) is small since the LS method finds R_d minimizing $|R_d(\lambda)|$ in the convex hull, but not with the second part, where the residual will be rich in the eigenvectors associated with the eigenvalues outside H_k. As the number of approximated eigenvalues k increasing,

the first part will be much closer to zero, but the second part keeps still large. This results in an enormous increase of restarted GMRES preconditioned vector norm. Meanwhile, when GMRES restarts with the combination of a number of eigenvectors, the convergence will be faster even if the residual is enormous, and the convergence of GMRES can still be significantly accelerated. The peaks shown in Fig. 2 for each time restart of UCGLE represent these enormous residuals. The l times repeat of r_d before applying to next time restart can still enlarge its norm, and the selection of l is important for the acceleration. In the example of Fig. 2, we conclude that if l is too large, the norm trends enormous, which slows down the speedup, if l is small, the acceleration may not be evident. In some situation, the preconditioned residual may be too large, it results that GMRES cannot converge enough before next restart. Hence the LS preconditioning can be applied each L times of GMRES restart, and the best l should be found.

3.3 Solving Linear Systems in Sequence by Recycling of Eigenvalues

After the analysis of relation between LS polynomial residual and the approximated eigenvalues, it is apparent that these dominant eigenvalues used by LS Component to accelerate the convergence, can be recycled and improved from the procedure of solving system with one RHS to another, which will introduce a potential continuous improvement for solving a long sequence of linear systems.

In order to solve the sequence of linear systems $Ax = b_t$ with $t \in 1,2,3,\cdots$. We enlarge the Krylov subspace size m_a inside ERAM Component to approximate more eigenvalues. Suppose that $(m_a)_1$ for the first system, the exact implementation of ERAM Component for $t \in 2,3,\cdots$ is shown in Equation (24), $(m_a)_t$ is equal to the sum of $(m_a)_{t-1}$ and a given constant a. And k_t, the number of eigenvalues computed by ERAM Component for $Ax = b_t$ can be described by a function f which maps the relation between $(m_a)_t$ and k_t. Obviously, $k_t \geq k_{t-1}$. The residual $(r_d)_t$ for each restart of $Ax = b_t$ with $t \in 2,3,\cdots$ is also given in (24).

$$\begin{cases} (m_a)_t = (m_a)_{t-1} + a \\ k_t = f(m_t) \\ (r_d)_t = \sum_{i=1}^{k_t} (R_d(\lambda_i))^l \rho_i u_i + \sum_{i=k_t+1}^{n} (R_d(\lambda_i))^l \rho_i u_i s \end{cases} \tag{24}$$

With the enlargement of ERAM Component Krylov subspace size, the more eigenvalues are calculated, then the first part of $(r_d)_t$ in Equation (24) are more important, and the more significant the acceleration will be. The continuous amelioration of convergence for solving linear systems in sequence can be gotten. With the changing of ERAM component Krylov subspace size, it may not be guaranteed to get the demanded eigenvalues in time for each restart of GMRES component if this size is too large comparing with GMRES Component Krylov subspace size. In order to improve the robustness of UCGLE, the previously calculated eigenvalues are kept in memory and updated if there come the new ones. These values in memory can be utilized in case that the failure of ERAM component when the parameters are too strict. In Equation (24), we did not define the upper limit for $(m_a)_t$, which depends on properties of operator matrices and P_g and P_a for GMRES and ERAM components.

For $t \in 2,3,\cdots$, since the eigenvalues calculated when solving $Ax = b_{t-1}$ are kept in memory, they can be used to construct an approximative solution for the current linear system $Ax = b_t$ through the LS polynomial method before its solve by GMRES. Obviously, this approximative solution can be used as a non-zero initial guess vector $(x_0)_t$ to solve $Ax = b_t$. It

will introduce an acceleration on the convergence for solving the linear systems in sequence. With the number of linear systems to be solved increasing, there will be more eigenvalues approximated, the initial guess vector constructed by LS component will be more accurate, and thus the speedup for solves from one to another can be still gotten. In fact, the impact of initial guess vector on the convergence is different from the restarted residual vector inside the iterative method, we propose a new parameter l'_t for the initial guess generation procedure which is different from the l in LS preconditioning part. The residual vector $(g_d)_t$ for $Ax = b_t$ with $t \in 2,3,\cdots$ is given in Equation (25), which is constructed by the k_{t-1} number of eigenvalues calculated when solving $Ax = b_{t-1}$.

$$(g_d)_t = \sum_{i=1}^{k_{t-1}} (R_d(\lambda_i))^{l'_t} \rho_i u_i + \sum_{i=k_{t-1}+1}^{n} (R_d(\lambda_i))^{l'_t} \rho_i u_i. \tag{25}$$

In this section, two parameters are added in order to solve linear systems in sequence using UCGLE, they are listed as below:

* $(m_a)_t$: ERAM Krylov subspace size for solving $Ax = b_t$
* l': times that LS polynomial applied for the generation of initial guess vector

It is predictable that this speedup for solving successive systems will stagnate after the optimized values of m_a and l' are found. It is useless to use ERAM Component to approximate the eigenvalues continuously. Thus P_a computing units allocated for ERAM Component can be redistributed to GMRES Component. It is expected to get an extra speedup on the performance with more computing resources.

The Algorithm 7 gives the procedure of UCGLE for solving a sequence of linear systems $Ax = b_t$ with $t \in 1,2,3,\cdots$. Initially, P_g and P_a are respectively set to be $proc_g$ and $proc_a$. If $t = 1$, UCGLE loads normally the three computing components with the ERAM component's Krylov subspace to be $(m_a)_1$, and the initial guess vector for GMRES component to be zero. For the solves of the successive linear systems, before the update of three

Algorithm 7 UCGLE for sequences of linear systems

for $(t \in (1,2,3,\cdots))$ **do**
  if $(t = 1)$ **then**
    set $P_g = proc_g$ and $P_a = proc_a$
    $LOADERAM(input : A, m_a, v, r, \varepsilon_a)$
    $LOADLS(input : A, b, d)$
    $LOADGMRES(input : A, m_g, x_0, b_1, \varepsilon_g, L, l, output : x_m)$
  else
    set $P_g = proc_g$ and $P_a = proc_a$
    $INITIAL\_GUESS(input : A, b_t, d, l'_t, output : g_{d_t})$
    update $LOADGMRES(input : A, m_g, g_{d_t}, b_t, \varepsilon_g, L, l, output : x_m)$
    update $(m_a)_{t-1}$ by $(m_a)_t$ in $LOADERAM(input : A, (m_a)_t, v, r, \varepsilon_a)$
    update $LOADLS(input : A, b_i, d)$
    if (optmized $(m_a)_{op}$ and l'_{op} found) **then**
      save the eigenvalues to *eigenvalues.bin*
      set $P_g = proc_g + proc_a - 1$ and $P_a = 1$
      $INITIAL\_GUESS(input : A, b_t, d, l'_{op}, output : g_{d_t})$ by loading *eigenvalues.bin*
      $LOADGMRES(input : A, m_g, g_{d_t}, b_t, \varepsilon_g, L, l, output : x_m)$
      replace *LOADERAM* by a simple useless function
      $LOADLS(input : A, b_i, d)$ by loading *eigenvalues.bin*
    end if
  end if
end for

components, a *INITIAL_GUESS* function is performed which is exactly the same as the LS component but with different parameter l'. The *INITIAL_GUESS* function will generate an initial guess vector g_{dt}. Inside GMRES component, the initial guess vector is updated by g_{dt} before the start of solves. And the Krylov subspace Size of ERAM Component is replaced by $(m_a)_t$. When the optimized values of $(m_a)_{op}$ and l'_{op} are found, the eigenvalues are kept into a local file *eigenvalue.bin*. The P_g and P_a are respectively updated as $proc_g + proc_a - 1$ and 1. The *INITIAL_GUESS* function executes by loading *eigenvalue.bin*. *LOADGMRES* is restarted with the redeployment of its data onto $proc_g + proc_a - 1$ computing units. *LS* executes also with *eigenvalue.bin*. The retainment of 1 computing unit for ERAM Component aims to ensure the distributed and parallel implementation of UCGLE with high fault tolerance. But the inside kernel of ERAM Component is replaced by a simple function with keeping the data sending and receiving functionalities.

4 Experiments of UCGLE for Sequences of Linear Systems

In this section, we evaluate the UCGLE for solving the sequences of linear systems on the supercomputer using different generated test matrices. UCGLE with or without initial guess vector generation is compared with conventional restarted GMRES with or without available preconditioners (Jacobi and SOR) in our implementations. The parallel performance on different homogeneous and heterogeneous platforms is presented in [30]. Thus this paper concentrates on the numerical performance of UCGLE for solving non-Hermitian linear systems in sequence, and the parallel performance comparison will not be discussed. Indeed, the performance of UCGLE can achieve further improvement with unique implementation targeting at different computer architectures, but this is not the purpose of this paper. *Unite and Conquer approach* (including UCGLE) is a particular programming model which introduces a better performance on top of classic solvers and makes them be more suitable for modern computers. It is fair to prove the benefits of *Unite and Conquer approach* by comparing it with the implementations of classic solvers based on the same basic operations (distribution of matrix across the cores, parallel sparse matrix-vector operation, the orthogonalization in Arnoldi reduction, etc.) without specific optimization for different platforms. In fact, if we optimize the parallel implementation of classic solvers and also the components (especially GMRES Component) in UCGLE at the same time, the benefits of UCGLE by reducing the global communications and promoting the asynchronization are still there.

4.1 Experimental Hardwares

UCGLE is implemented on the supercomputer *Tianhe-2*, installed at the National Super Computer Center in Guangzhou of China. It is a heterogeneous system made of Intel Xeon CPUs and Matrix2000, with 16000 compute nodes in total. Each node composes 2 Intel Ivy Bridge 12 cores @ 2.2 GHz. In this paper, we did not test UCGLE with co-processor Matrix2000 on *Tianhe-2* since our implementation does not support it with good performance.

4.2 Large-scale Test Matrices Generation with Given Spectra

In order to test UCGLE with matrices of high dimensions, we use our Scalable Matrix Generator with Given Spectra (SMG2S) [29, 31] to generate different test matrices. SMG2S is

an open source package implemented and optimized using MPI and C++, which allows generating efficiently large-scale test matrices with customized eigenvalues by users to evaluate the impacts of spectra on the covergence of linear solvers on the large-scale platforms. Fig. 3 gives an example of SMG2S to generate test matrix. SMG2S has good scalability and acceptable accuracy to keep the given spectra. One more important benefit, since the matrices are generated by SMG2S in parallel, the data are already allocated on different processes. These distributed data can be used directly for the users to efficiently evaluate the numerical methods without concerning the I/O operation.

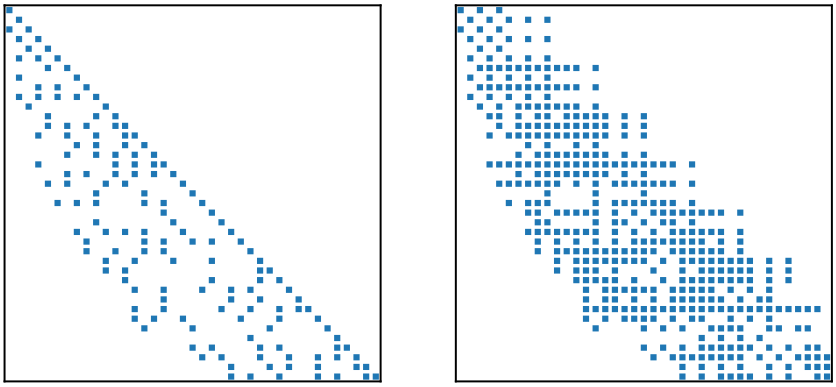

Fig. 3 Spasity Pattern of Matrix Generated by SMG2S. [31]

4.3 Experimental Results

We evaluate UCGLE for solving sequences of linear systems using three test matrices of size 1.572×10^7 generated by SMG2S with different given spectra. They are respectively denoted as $Mat1$, $Mat2$ and $Mat3$. The different right-hand sides of these sequent linear systems are generated at random. The parameter l for all tests using UCGLE keeps the same as 10. The numbers of process for GMRES and ERAM Components in UCGLE are respectively 768 and 384. Five methods are compared in the experiments, and the notations of different methods are given as below:

- GMRES: classic restarted GMRES;
- GMRES+SOR or SOR: GMRES with SOR preconditioner;
- GMRES+Jacobi or Jacobi: GMRES with Jacobi preconditioner;
- UCGLE without initial guess: UCGLE without using previously obtained eigenvalues to generate an initial guess vector for the next system by LS polynomial method;
- UCGLE with initial guess: UCGLE using previously obtained eigenvalues to generate an initial guess vector for the next systems by LS polynomial method.

ERAM and LS components in UCGLE demand additional computing units. It is unfair to test only the conventional methods of their numbers of CPUs equal to the number of GMRES components in UCGLE. Therefore, experiments have also been tested that the numbers of computing units of the classical iterative method are equal to the total CPU in

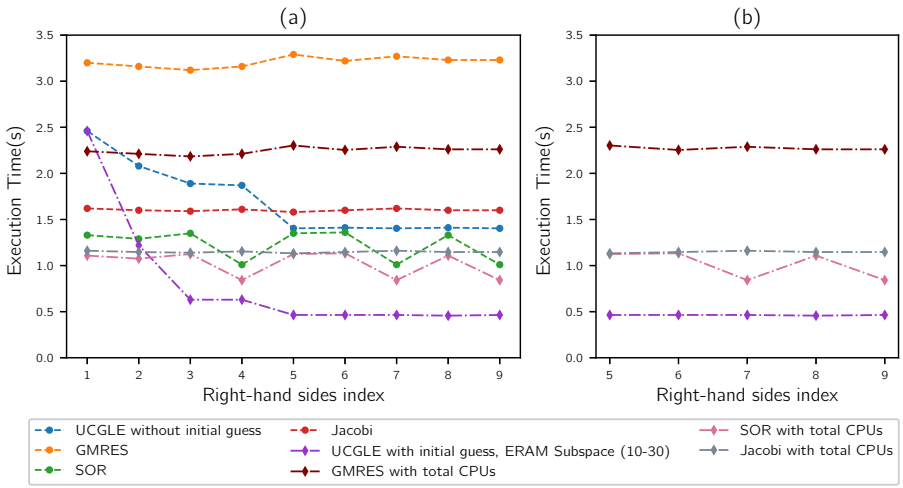

Fig. 4 *Mat*1: time comparison for solving a sequence of linear systems. (a) shows the solution time for 9 sequent linear systems; (b) shows the cases extracted from (a) after the good selection of parameters in UCGLE.

UCGLE (hence, the CPUs for GMRES and ERAM components are included). In the captions of figures, a given method "with total CPUs" means that its number of CPUs equals to the total CPU in UCGLE. The time comparisons for solving nine sequent linear systems of *Mat*1, *Mat*2 and *Mat*3 are given respectively in Fig. 4 (a), Fig. 5 (a) and Fig. 6 (a), the comparison of the number of iteration for convergence are respectively given in Table 1, Table 2 and Table 3. From these three tables, we conclude that UCGLE has the ability to speed up the convergence to solve linear systems with test matrices comparing with the conventional methods. The generation of initial vectors using the eigenvalues for subsequent linear systems can still speed up the convergence over the UCGLE without initial guess.

Table 1 *Mat*1: iterative step comparison for solving a sequence of linear systems.

Method	1	2	3	4	5	6	7	8	9
GMRES	509	505	501	505	527	510	523	516	518
GMRES+SOR	169	165	172	130	172	173	130	170	130
GMRES+Jacobi	274	273	270	276	269	274	280	276	273
UCGLE w/o initial guess	120	90	90	90	90	90	90	90	90
UCGLE with initial guess	120	36	35	35	36	36	36	33	35

For the tests of *Mat*1, the GMRES restart size is 30, the Krylov subspace of ERAM Component for solving the first three linear systems are respectively 10, 20 and 30, the size of this subspace of ERAM for the remaining systems keeps being 30. For the tests of *Mat*2, the GMRES restart size is 300, the Krylov subspace of ERAM Component for solving the first three linear systems are respectively 100, 150 and 200, the size of this subspace of ERAM for the remaining systems keeps being 200. For the cases that UCGLE

with initial guess, the parameter l' for $Mat1$ and $Mat2$ keeps 30. With the augmentation of the size of ERAM Krylov subspace, there will be more eigenvalues to be approximated, and we find that there is acceleration with the accumulation of more eigenvalues for both the case UCGLE with and without initial guess. The influence of subspace of ERAM can be found through the curves of UCGLE with/without initial guess in Fig. 4 (a) and Fig. 5 (a). However, it is not practical to enlarge too much the Krylov subspace of ERAM to approximated more eigenvalues, since if it is too large, it takes too much time by ERAM, LS Component cannot receive the eigenvalues in time, thus it will be difficult for the GMRES Component to perform the LS preconditioning for it each time restart.

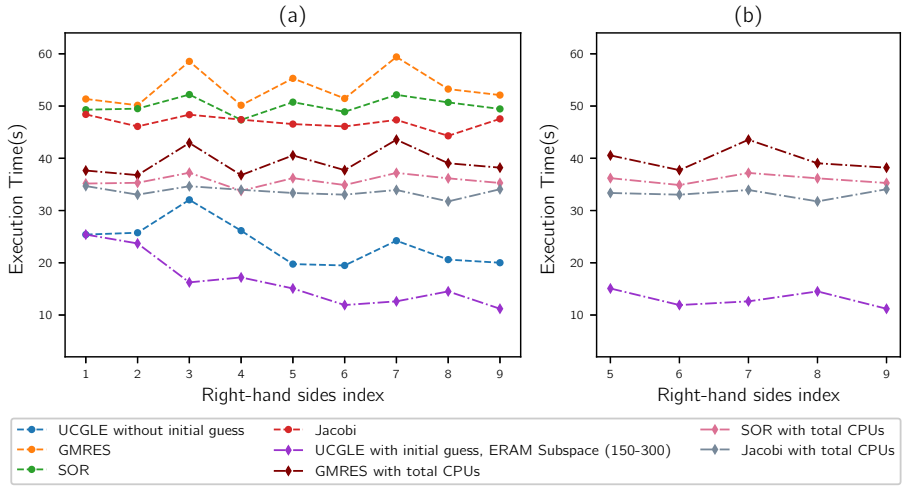

Fig. 5 $Mat2$: time comparison for solving a sequence of linear systems. (a) shows the solution time for 9 sequent linear systems; (b) shows the cases extracted from (a) after the good selection of parameters in UCGLE.

Table 2 $Mat2$: iterative step comparison for solving a sequence of linear systems.

Method	1	2	3	4	5	6	7	8	9
GMRES	1316	1277	1460	1278	1409	1325	1472	1369	1342
GMRES+SOR	1197	1219	1336	1173	1290	1194	1335	1289	1213
GMRES+Jacobi	1278	1185	1283	1220	1191	1184	1218	1159	1239
UCGLE w/o initial guess	666	671	831	689	701	685	837	736	714
UCGLE with initial guess	666	595	470	491	544	464	485	532	440

For the tests of $Mat3$, the GMRES restart size is 150, and the Krylov subspace size of ERAM Components keeps the same to be 200. Meanwhile, for the 2nd, 3rd and 4th linear systems, the parameter l' of initial guess are respectively 20, 30 and 40, for the remaining linear systems, this parameter keeps 40. For the solving the linear systems by UCGLE with

initial guess, we can find that with the augmentation of l', the iteration numbers for the first four linear systems decrease quickly from 360 to 283 with approximately 1.3× speedup. For *Mat*3, SOR preconditioned GMRES is already good, but UCGLE with initial guess has still about 2.2× speedup of convergence. Even in the case that the computing unit number of SOR preconditioned GMRES equals to the total number of UCGLE, UCGLE with initial guess can achieve 2.2× of execution time speedup. The augmentation of the parameter l' has a strong impact on the convergence. Since the *Mat*3 is generated with the clustered eigenvalues which are randomly distributed inside a fixed region of the real-imaginary plan, if l' is larger, it can be seen as there are much more eigenvalues generated, even they are not very accurate compared with the real ones. The inaccuracy of eigenvalues can result in the enlargement of the norm in Equation (24), but it can still very quickly converge. It is effective to generate an initial guess vector with very large l', but not the same case for the parameter l inside each preconditioning, since the too many times of repeats for each time restart will amplify quicky this inaccuracy of norm, and it is easy to result in the difficulties of convergence.

In order to use UCGLE for solving a large number of linear systems in sequence, it is necessary to choose the suitable parameters by the evaluation of a small number of sequent linear systems. After the selection of parameters, we compare the best cases of each method with the least time consumption. The results for three test matrices are shown in Fig. 4 (b), Fig. 5 (b) and Fig. 6 (b). We conclude that for *Mat*1, UCGLE with initial guess has about 4.4× for the acceleration of convergence and 1.7× for the speedup of execution time. For *Mat*2, it has about 4.3× acceleration for the convergence and 2.6× for the speedup of time. For *Mat*3, it has about 3.2× acceleration for the convergence and 2.0× for the speedup of execution time.

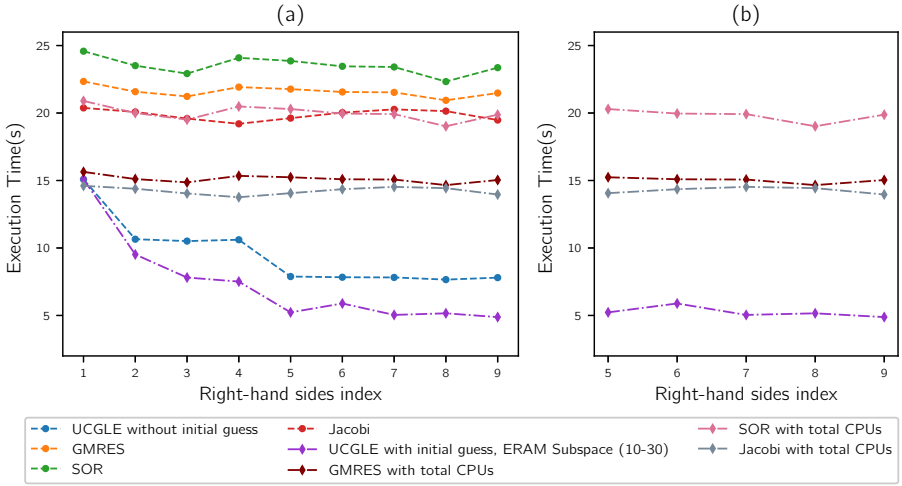

Fig. 6 *Mat*3: time comparison for solving a sequence of linear systems. (a) shows the solution time for 9 sequent linear systems; (b) shows the cases extracted from (a) after the good selection of parameters in UCGLE.

In conclusion, the UCGLE, especially it with the recycling eigenvalues to generate initial guess vector using the eigenvalues, can significantly accelerate the convergence and reduce

Table 3 *Mat*3: iterative step comparison for solving a sequence of linear systems.

Method	1	2	3	4	5	6	7	8	9
GMRES	914	912	892	885	895	905	911	892	904
GMRES+SOR	895	871	856	885	879	870	868	838	868
GMRES+Jacobi	894	888	875	864	876	887	892	888	872
UCGLE w/o initial guess	673	364	355	360	367	363	363	351	364
UCGLE with initial guess	673	396	291	283	339	338	274	279	267

the time consumption for solving a sequence of linear systems. However, the time employed by the LS iterative recurrence, especially a small number of Sparse Matrix-Vector Product operations inside makes the time speedup not consistent with the convergence speedup. For example, in Table 3, UCGLE with initial guess has almost $3.0\times$ acceleration on the convergence over the classic GMRES for solving the 3rd linear system. But it has only about $2.0\times$ acceleration on the performance in the case that the classic GMRES and GMRES Component inside UCGLE have the same number of computing units. It is caused by the recurrence of LS iterations to perform the preconditioning on GMRES Component after receiving the parameters from LS Component. Nevertheless, UCGLE is more efficient to solve the sequences of linear systems, and with its distributed and parallel communication framework, it is a good candidate for solving non-Hermitian linear systems in sequence on much larger machines.

5 Conclusion and Perspective

This paper proposes an extended version of the distributed parallel method UCGLE, which is used to solve a large number of linear systems with unique matrices and different right sides on a large platform. UCGLE method was proposed to solve large-scale linear systems on modern computing platforms, which is able to minimize the global communication, cover the synchronous points in the parallel implementation, improve the fault tolerance and reusability and speed up the convergence. In this paper, It is proved this developed variant of UCGLE method can solve the linear systems with special spectral distribution in sequence more effectively than several preconditioned iterative methods. The recycling of a small group of dominant eigenvalues and generating initial guess vector using them by Least Squares polynomial method has a significant impact on the performance improvement for solving continuous linear systems.

Various parameters have an impact on the convergence. Thus, the auto-tuning scheme is required in the future work, where the systems can select different Krylov subspace dimensions, numbers of eigenvalues to be computed, degrees of Least Squares polynomial according to different linear systems and cluster architectures.

References

1. Abdel-Rehim AM, Morgan RB, Wilcox W (2014) Improved seed methods for symmetric positive definite linear equations with multiple right-hand sides. Numerical Linear Algebra with Applications 21(3):453–471

2. Arnoldi WE (1951) The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of applied mathematics 9(1):17–29
3. Baker AH, Dennis JM, Jessup ER (2006) On improving linear solver performance: A block variant of gmres. SIAM Journal on Scientific Computing 27(5):1608–1626
4. Bellavia S, Morini B (2001) A globally convergent newton-gmres subspace method for systems of nonlinear equations. SIAM Journal on Scientific Computing 23(3):940–960
5. Benner P, Feng L (2011) Recycling krylov subspaces for solving linear systems with successively changing right-hand sides arising in model reduction. In: Model Reduction for Circuit Simulation, Springer, pp 125–140
6. Brown PN, Saad Y (1990) Hybrid krylov methods for nonlinear systems of equations. SIAM Journal on Scientific and Statistical Computing 11(3):450–481
7. Calvetti D, Reichel L (1994) Application of a block modified chebyshev algorithm to the iterative solution of symmetric linear systems with multiple right hand side vectors. Numerische Mathematik 68(1):3–16
8. Craig RR, Hale AL (1988) Block-krylov component synthesis method for structural model reduction. Journal of Guidance, Control, and Dynamics 11(6):562–570
9. Dongarra J, Hittinger J, Bell J, Chacon L, Falgout R, Heroux M, Hovland P, Ng E, Webster C, Wild S (2014) Applied mathematics research for exascale computing. Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA
10. Emad N, Petiton S (2016) Unite and conquer approach for high scale numerical computing. Journal of Computational Science 14:5–14
11. Essai A, Bergére G, Petiton SG (1999) Heterogeneous parallel hybrid gmres/ls-arnoldi method. In: PPSC
12. Flueck AJ, Chiang HD (1998) Solving the nonlinear power flow equations with an inexact newton method using gmres. IEEE Transactions on Power Systems 13(2):267–273
13. Gu GD (2002) A seed method for solving nonsymmetric linear systems with multiple right-hand sides. International journal of computer mathematics 79(3):307–326
14. Gullerud AS, Dodds Jr RH (2001) Mpi-based implementation of a pcg solver using an ebe architecture and preconditioner for implicit, 3-d finite element analysis. Computers & Structures 79(5):553–575
15. Gutknecht MH (2006) Block krylov space methods for linear systems with multiple right-hand sides: an introduction
16. He H, Bergère G, Petiton S (2006) A hybrid gmres/ls-arnoldi method to accelerate the parallel solution of linear systems. Computers & Mathematics with Applications 51(11):1647–1662
17. Jolivet P, Tournier PH (2016) Block iterative methods and recycling for improved scalability of linear solvers. In: SC16-International Conference for High Performance Computing, Networking, Storage and Analysis
18. Kershaw DS (1978) The incomplete choleskyconjugate gradient method for the iterative solution of systems of linear equations. Journal of Computational Physics 26(1):43–65
19. Kilmer ME, De Sturler E (2006) Recycling subspace information for diffuse optical tomography. SIAM Journal on Scientific Computing 27(6):2140–2166
20. Knoll DA, Keyes DE (2004) Jacobian-free newton–krylov methods: a survey of approaches and applications. Journal of Computational Physics 193(2):357–397
21. Papadrakakis M, Smerou S (1990) A new implementation of the lanczos method in linear problems. International Journal for Numerical Methods in Engineering 29(1):141–159

22. Parks ML, De Sturler E, Mackey G, Johnson DD, Maiti S (2006) Recycling krylov subspaces for sequences of linear systems. SIAM Journal on Scientific Computing 28(5):1651–1674
23. Saad Y (1987) Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems. SIAM Journal on Numerical Analysis 24(1):155–169
24. Saad Y (1987) On the lanczos method for solving symmetric linear systems with several right-hand sides. Mathematics of computation 48(178):651–662
25. Saad Y (2011) Numerical Methods for Large Eigenvalue Problems: Revised Edition. SIAM
26. Saad Y, Schultz MH (1986) Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on scientific and statistical computing 7(3):856–869
27. Simoncini V, Gallopoulos E (1995) An iterative method for nonsymmetric systems with multiple right-hand sides. SIAM Journal on Scientific Computing 16(4):917–933
28. Smith CF, Peterson AF, Mittra R (1989) A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields. IEEE Transactions on Antennas and Propagation 37(11):1490–1493
29. WU X (2018) SMG2S Manual v1.0. Technical report, Maison de la Simulation
30. Wu X, Petiton SG (2018) A distributed and parallel asynchronous unite and conquer method to solve large scale non-hermitian linear systems. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, ACM, New York, NY, USA, HPC Asia 2018, pp 36–46, DOI 10.1145/3149457.3154481
31. WU X, Petiton S, Lu Y (2018) A Parallel Generator of Non-Hermitian Matrices computed from Given Spectra. In: VECPAR 2018: 13th International Meeting on High Performance Computing for Computational Science, So Paulo, Brazil
32. Ye Z, Zhu Z, Phillips JR (2008) Generalized krylov recycling methods for solution of multiple related linear equation systems in electromagnetic analysis. In: Proceedings of the 45th annual Design Automation Conference, ACM, pp 682–687
33. Zhang Y, Bergere G, Petiton S (2008) Large scale parallel hybrid gmres method for the linear system on grid system. In: Parallel and Distributed Computing, 2008. ISPDC'08. International Symposium on, IEEE, pp 244–249